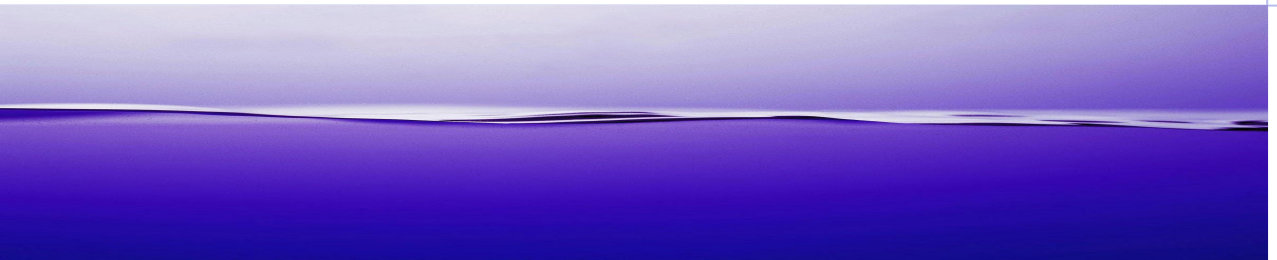
Implementing Missing C and C++ Language Features in Clang

Muhammad Bassiouni
C++ Alliance Fellowship 2026
Mentors: Vassil Vassilev, Aaron Jomy





About Me

01 **Bachelor of Artificial Intelligence**

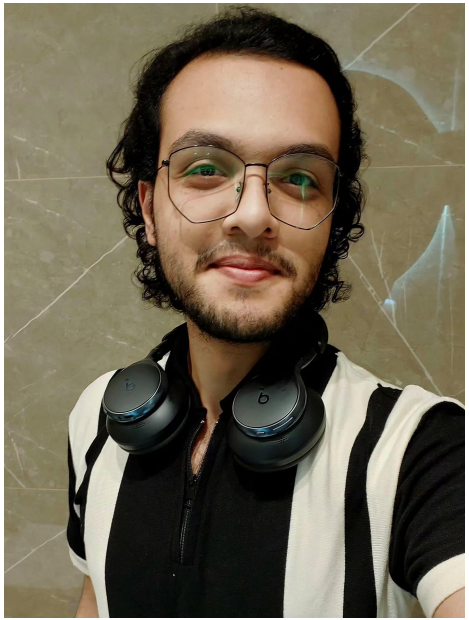
Class of 2025

02 **Maintainer of LLVM libc**

The LLVM C standard library

03 **GSoC 2026 mentor for LLVM**

LLVM libc and compiler-rt projects



The Fellowship at a Glance

1

C2y N3356
Declarations in if / switch conditions

Landed upstream

2

C++20 P1814R0
CTAD for alias templates

Blocked: open core issue

3

C++26 constexpr math
constexpr <cmath> in Clang

Blocked: pending design

4

C++29 P3668R4
Defaulted postfix ++ / --

PR filed

One paper landed, two were blocked, and the last one has a PR filed.

C2y N3356

Declarations in if and switch conditions

N3356: What the Paper Adds

01

Adds a single declaration

A declaration is allowed in the condition of if and switch.

02

Similar to C++, but not the same

The first clause is a declaration and can't be empty; the second clause is only an expression.

03

More declaration specifiers

Because the first clause is a declaration, attributes and specifiers come with it.

```
1  if (bool x = true; x) {}
2  if (bool x = false) return x;
3  if ([[maybe_unused]] bool x = true) {}
4  if (int x = 3; x > 0) {}
5  if (struct A { int x; } a = {.x = 1}; a.x) {}
6  if (int arr[] = {1,2,3}; arr[1]) {}
7  if (static_assert(true); true) {}
8  if ([[]]; true) {}
9  if (auto x __attribute__((deprecated)) = 3) {}
10 if (__extension__ [[maybe_unused]] int x = 3;
11     x > 0) {}
12
13 switch (int x = 1) {}
14 switch (auto x = 3) {default: x += 1;}
```

How it was built: *reuse the existing C++ code paths, then specialize them for C2y*

Existing C++17 path

```
if (init; cond)
switch (init; cond)
```

→

Specialized for C2y

First clause must be a declaration
Second clause is an expression only
C attributes, specifiers, `__extension__`

→

C2y N3356 in Clang

Every C mode, C89 to C2y
with compatibility diagnostics

N3356: What Landed in Clang

01

Merged upstream

Complete per the paper wording, part of Clang 24 (commits 226b0b5 and 1387002).

02

Available back to C89

Works in every C mode Clang supports, with compatibility warnings and diagnostics.

03

__extension__ in C and C++

Silences extension diagnostics for the rest of the condition. It wasn't available in C++ before.

Did not make it

GNU attribute declarations like (__attribute__((...)); true). GCC doesn't follow the paper wording here yet, so we couldn't decide. C23-style attribute declarations did make it.

Side effect

Commit 1387002 introduced a new way to handle compatibility diagnostics.

How It Impacted Clang

C89 - C2y

the new syntax works in every C mode Clang supports

1

user issues reported since the merge, despite heavy parser changes

0

regressions reported by Ezzeldin: the new code is as good as the old

Also brought __extension__ to C++ conditions and a new approach to compatibility diagnostics.

What Got Blocked, and Why

CTAD for alias templates and constexpr math

What Is CTAD?

Class template argument deduction (C++17) lets the compiler work out a class template's arguments from its initializer, as it already did for function templates.

Before C++17: spell it out

```
1  std::pair<int, double> p(1, 2.0);
2  auto q = std::make_pair(1, 2.0);
3  std::vector<int> v = {1, 2, 3};
4  std::lock_guard<std::mutex> lk(m);
```

With CTAD: let it deduce

```
1  std::pair p(1, 2.0);        // <int, double>
2  auto q = std::pair(1, 2.0);
3  std::vector v = {1, 2, 3}; // <int>
4  std::lock_guard lk(m);      // <std::mutex>
```

01

Guides from constructors

Each constructor becomes an implicit deduction guide: a function template returning the class.

02

Pick one like a call

The initializer goes through overload resolution against those guides.

03

User-written guides

Libraries add rules constructors can't express, e.g. vector(It, It) → vector<value_type>.

P1814R0: CTAD for Alias Templates

Extends class template argument deduction (CTAD) to alias templates in C++20.

```cpp
template<typename T> struct A { A(T) {} };
template<typename T> using Proxy = T;
template<typename T> using C = Proxy<A<T>>;

C test{42};  // deduce C<int>
```

Where Clang stood

- Basic implementation, not stable
- Rejects valid code or crashes in some cases
- Many reported issues needing triage

01

The PR got blocked

Debate on equivalence of alias templates; only GCC implements it.

02

Open core issue: CWG2186

Fixing it needs new wording for the committee.

03

Not a priority

Wording wouldn't be discussed soon, so it didn't fit the fellowship schedule.

Pivot: C++23/26 constexpr Math

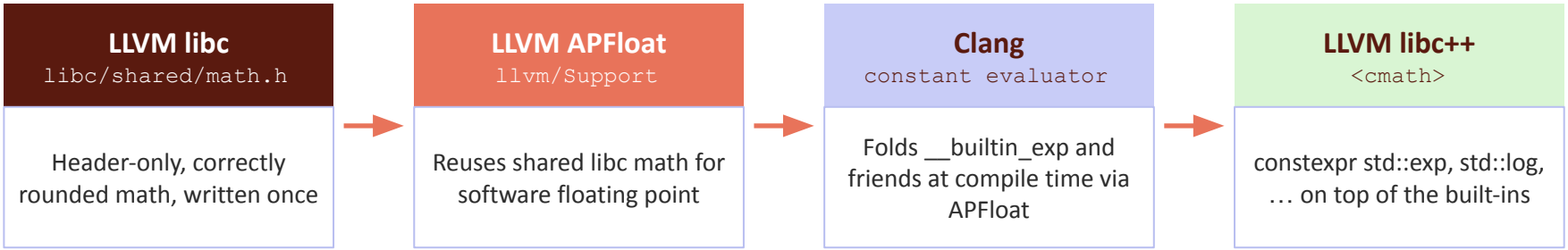
01

A natural next step

Half of the work is in Clang built-ins, and I had already done the LLVM libc side.

Hand-in-Hand++ Hand-in-Hand started by sharing LLVM libc internals with libc++. Hand-in-Hand++ extends it to the whole LLVM tree: `libc/shared/` exposes header-only, correctly rounded math that LLVM, Clang and libc++ can include, so every layer uses one implementation instead of rewriting it.

Implementing constexpr math spans four LLVM projects



constexpr Math: Why I Pivoted

02

Waiting on __builtin_exp

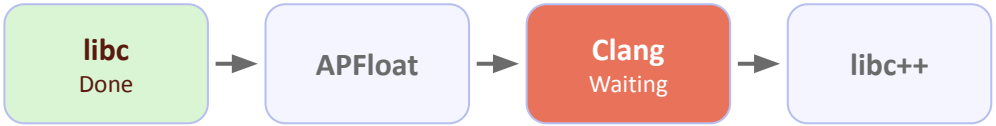
An open PR makes __builtin_exp constexpr. Maintainers first need to agree on the approach.

03

Too slow for the fellowship

With the design still open, I picked another paper to implement.

Where the chain stands



The libc side is in place; the chain is held up at Clang until the `__builtin_exp` approach is agreed on.



P3668R4: Defaulted Postfix ++ and --

```
struct Counter {
  int v = 0;
  Counter& operator++() { ++v; return *this; }
  Counter operator++(int) = default;
  // same as:
  //   Counter tmp = *this;
  //   ++*this; return tmp;
};
```

The postfix form is generated from the prefix one.

01 **Filed a PR**

Implements the paper wording in Clang.

02 **Ported down to C++03**

Had to account for language features like constexpr and other qualifiers not being available.

03 **Reviewed upstream**

Worked with reviewers on the implementation.

How I Used AI

01

Navigating Clang's code

Explaining unfamiliar parser and Sema code paths, and finding where a change should go in a large codebase.

02

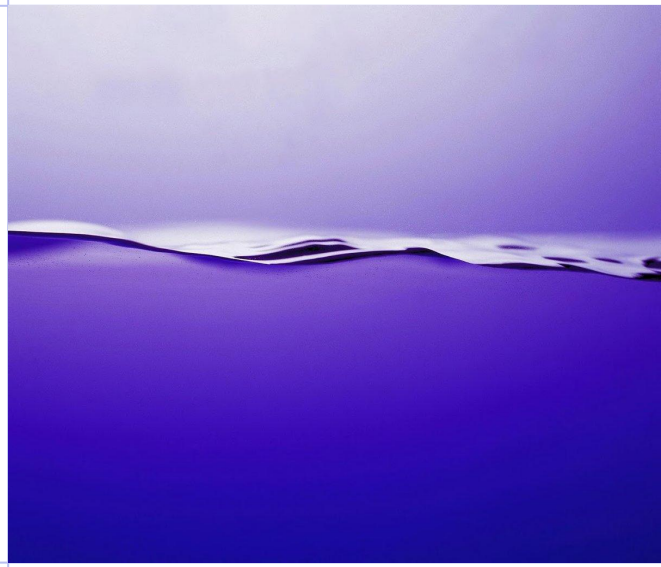
Writing tests

Drafting test cases and listing edge cases to cover, such as attributes, __extension__ and the different C modes.

I used AI as an assistant for exploring the code and drafting tests.

Thank You

Questions and feedback are welcome!



Find me at
muhammad.m.bassiouni@gmail.com
github.com/bassiounix
linkedin.com/in/bassiounix



